Konzeption eines SOA-Repository mit Analysefähigkeiten

Thomas Bauer [a], Stephan Buchwald [b], Julian Tiedeken [c], Manfred Reichert [c]

[a] Hochschule Neu-Ulm
Fakultät
Informationsmanagement
Wileystr. 1
89231 Neu-Ulm
thomas.bauer@
hs-neu-ulm.de

[b] T-Systems International
Delivery Unit Automotive
& Manufacturing Industries
Olgastr. 63
89073 Ulm
stephan.buchwald@
t-systems.com

[c] Universität Ulm
Institut für Datenbanken
und Informationssysteme
James-Franck-Ring
89081 Ulm
{manfred.reichert, julian.
tiedeken}@uni-ulm.de

Abstract: In einer SOA werden bereitgestellte Services von Servicenutzern, etwa fremden IT-Systemen konsumiert. Eine Serviceänderung bzw. -abschaltung kann solche Servicenutzer schwer beeinträchtigen. Deshalb müssen diese vor einem solchen Eingriff ermittelt werden, wozu das SOA-Repository genutzt werden kann. Allerdings ist bei umfangreichen SOA-Repositories schwer erkennbar, welche IT-Systeme (evtl. indirekt) in welchem Zeitraum betroffen sein werden. Da diese Problemstellung in der bisherigen SOA-Literatur nicht betrachtet wird, stellen wir ein Lösungskonzept vor. Es ermöglicht automatisierte Analysen, um in den Daten enthaltene Problemsituationen zu identifizieren (Ist-Analysen), und auch, um Auswirkungen zukünftig durchzuführender Serviceänderungen zu simulieren (Planspiele).

1 Einleitung

Service-orientierte Architekturen (SOA) führen zu einer Erhöhung der Flexibilität. So ermöglicht beispielsweise die lose Kopplung beim Service-Aufruf, schnell zur Verwendung eines anderen Services zu wechseln. Dadurch dass ein Proxy (Enterprise Service Bus, ESB) zwischen die aufrufende Applikation und den verwendeten Service geschaltet wird, muss für diesen Wechsel lediglich der Proxy verändert werden, jedoch nicht die konsumierende Applikation selbst. Wir nutzen im Projekt ENPROSO[1] zahlreiche solcher Mechanismen zur Erhöhung der Flexibilität [BBR11, Bu12].

Die hohe Flexibilität bei Service-Aufrufen kann jedoch zu einer Starre beim Betrieb der Services führen: So werden manche Veränderungen an bereitgestellten (d.h. betriebenen) Services erschwert. Dies gilt insb. für die Abschaltung von Services, da dies die Funktionsfähigkeit der Servicenutzer (Consumer) beeinträchtigen kann. Ein solcher Fall muss unbedingt vermieden werden, weil selbst ein Ausfall einzelner Funktionalitäten der Service-nutzenden Applikationen meist nicht akzeptabel ist. Andererseits verursacht ein lange andauernder Weiterbetrieb vieler veralteter Services hohe Kosten. Das bedeutet, dass ein Konzept entwickelt werden muss, um ungenutzte Services zuverlässig identifizieren und dann abschalten zu können (Service Sun-down). Solche Services können

[1] Enhanced Process Management by Service Orientation, durchgeführt bei und finanziert von der Daimler AG.

mit Informationen aus dem SOA-Repository identifiziert werden. Als Grundlage hierfür haben wir im Projekt ENPROSO ein sehr umfassendes Metamodell für ein SOA-Repository konzipiert [BTBR10, Bu12, Ti10]. Dieses stellt detaillierte Information zu Service-Versionen und -Installationen ebenso bereit wie zu Service-Nutzungsvereinbarungen (Contracts). Mit diesen und weiteren Informationen ist erkennbar, welche konkreten Service-Versionen und -Installationen tatsächlich genutzt werden, bis wann dies erfolgt und welche verzichtbar sind bzw. in Zukunft abgeschaltet werden können. Damit kann der Anpassungsaufwand für die konsumierende Applikation abgeschätzt werden.

Hierdurch ergibt sich ein umfangreiches SOA-Repository mit vielen verschiedenen Objekttypen und zudem jeweils vielen -Instanzen. Kritisch daran ist, dass zu erwartende Problemsituationen für menschliche Benutzer aufgrund der Datenmenge kaum mehr direkt erkennbar sind. Deshalb werden automatisierte Analysemöglichkeiten benötigt: Basierend auf den existierenden SOA-Repository-Daten müssen insbesondere zukünftig auftretende Problemsituationen entdeckt werden. Da es zu solchen automatischen Analyseverfahren bisher keine SOA-Literatur gibt, stellen wir in diesem Beitrag entsprechende Verfahren vor, um z.B. Services zu finden, deren Abschaltung vorgesehen ist, bevor ihre Nutzungsvereinbarung ausläuft.

Als Erweiterung zur Erkennung von solchen – aufgrund der existierenden Daten bereits „vorprogrammierten“ – Problemsituationen (d.h. Ist-Analysen) sollte ein SOA-Repository auch Planspiele (What-if-Analysen) ermöglichen: Um einen geeigneten Zeitpunkt für eine Service-Abschaltung wählen zu können, müssen die jeweiligen Folgen analysierbar sein. Dazu müssen mehrere Abschaltzeitpunkte betrachtet werden, wobei jeweils die betroffenen Service-Consumer und Lösungsalternativen identifiziert werden (Impact-Analysen). Da die Bewertung mehrerer potentieller Abschaltzeitpunkte basierend auf den SOA-Repository-Daten einen großen Aufwand verursachen würde, stellen wir in diesem Beitrag (ebenfalls originäre) Verfahren für automatisierte Impact-Analysen vor.

Als Grundlage für die Analysen stellen wir in Abschnitt 2 unser SOA-Repository-Metamodell auszugsweise vor. Abschnitt 3 beschreibt exemplarisch einige Analysen basierend auf aktuell existierenden Repository-Daten (Ist-Analysen). Planspiele und die zugehörigen Impact-Analysen werden in Abschnitt 4 betrachtet. Eine Diskussion des aktuellen Standes der Literatur sowie existierende Plattformen für SOA-Repositories findet sich in Abschnitt 5. Der Beitrag schließt mit einer Zusammenfassung.

2 Grundlagen: Metamodell des SOA-Repository

Für die Ermittlung der Anforderungen an ein SOA-Repository wurden in mehreren Geschäftsbereichen der Daimler AG Anforderungsanalysen durchgeführt [BTBR10, Bu12]. Hierbei ergab sich außer dem Bedarf an Analysen (siehe Abschnitte 3 und 4) auch die Notwendigkeit zur Speicherung diverser Objekt- und Beziehungstypen in einem SOA-Repository. Das entsprechende Metamodell zur Datenspeicherung ist in Abbildung 1 als Entity-Relationship-Modell dargestellt. Zur Beschreibung von Kardinalitäten wird die (min,max)-Notation verwendet [Ab74].

Das Metamodell umfasst sowohl fachliche (linke Hälfte) wie auch technische (rechte Hälfte) Objekttypen und ist auch bzgl. der konkreten Objekttypen sehr umfangreich. So werden u.A. fachliche und technische *DataObjectVersions*[2] berücksichtigt, weil auch diese für den Anwender z.B. beim Suchen bzw. Browsen nach verwendbaren Services hilfreich sein können. Ziel dieses Metamodells ist, sehr viele der häufig in einer SOA benötigten Informationen abzubilden, jedoch können abhängig vom jeweiligen Unternehmen und seiner SOA-Philosophie zusätzliche Objekt- und Beziehungstypen erforderlich werden bzw. dann irrelevante entfallen. Im Folgenden erläutern wir aus Platzgründen nur die für das Verständnis dieses Beitrags erforderlichen Objekt- und Beziehungstypen. Für eine Erläuterung der anderen und auch für eine Diskussion der jeweils relevanten Attribute wird auf [Bu12, Ti10] verwiesen.

Zentrales Objekt in einer SOA ist der *Service* und findet sich dementsprechend auch im Metamodell. Allerdings ist ein Service nur eine „Bündelung" von unterschiedlichen Service-Versionen, da sich ein „konkreter Service" im Laufe der Zeit weiterentwickelt. Die entstehenden Service-Versionen können unterschiedliche Ein-/Ausgabeparameter oder sogar Operationen haben, weshalb sie für den Consumer inkompatibel sind. Wir unterscheiden zudem in fachliche (*BusinessServiceVersion*, verbunden mit Beziehungstyp *isBusinessServiceVersion*) und technische Service-Versionen (*TechnicalServiceVersion*, verbunden mit *isTechnicalServiceVersion*). Fachlich wird ein Service so beschrieben, dass Anwender aus Fachabteilungen und Manager den Zweck und groben Inhalt des Service verstehen und bewerten können. Erst nachdem eine Realisierung der fachlichen Service-Version beschlossen wurde, entsteht eine technische Service-Version inkl. einer detaillierten Spezifikation u.a. der Schnittstelle (WSDL). Eine fachliche Service-Version kann auf unterschiedliche Arten umgesetzt werden – auch mehrfach – wobei sich die zugehörigen technischen Service-Versionen z.B. in den Datentypen der Parameter oder der Anzahl der Operationen unterscheiden können. Da technische Service-Versionen inhaltlich dieselbe Aufgabe erledigen, sich aber in technischen Details unterscheiden, werden mehrere von ihnen mittels *hasTechnServiceVersImplementation* derselben fachlichen Service-Version zugeordnet.

Nach Implementierung und Rollout (Deployment) einer technischen Service-Version entsteht eine Service-Installation (*ServiceInstall*), die ihr zugeordnet wird. Da die Service-Installation von einem konkreten IT-System bereitgestellt wird, ist sie zudem mit der Beziehung *providesServiceInstall* genau diesem *System* zugeordnet. Die Beziehung *hasProvider* beschreibt etwas „ungenauer", dass das IT-System diesen Service bereitstellt, wobei ein Service i.Allg. von unterschiedlichen IT-Systemen angeboten werden und mehrere Installationen besitzen kann, die evtl. zu verschiedenen Service-Versionen gehören.

Soll ein Service tatsächlich verwendet werden, so muss zwischen dem Anbieter und dem Consumer ein Nutzungsvertrag (*Contract*) geschlossen werden. Die Zuordnung des Contract zu dem Consumer erfolgt mittels der Beziehung *hasContract*. Die Information zu diesem Contract entsteht typischerweise sukzessive, so dass zuerst nur festgelegt wird, welche fachliche Service-Version genutzt werden soll (Zuordnung mittels *has-*

[2] Versionen von Datentypen, die in Service-Operationen als Ein- oder Ausgabeparameter verwendet werden.

BusinServiceVersConsumer). Zu diesem Zeitpunkt muss noch gar keine technische Service-Version hierzu existieren. Wird dann auch eine solche ausgewählt, so wird der Contract ergänzt (evtl. auch um technische Service-Level-Agreements) und die Zuordnung zum Contract erfolgt mittels *hasTechnServiceVersConsumer*.

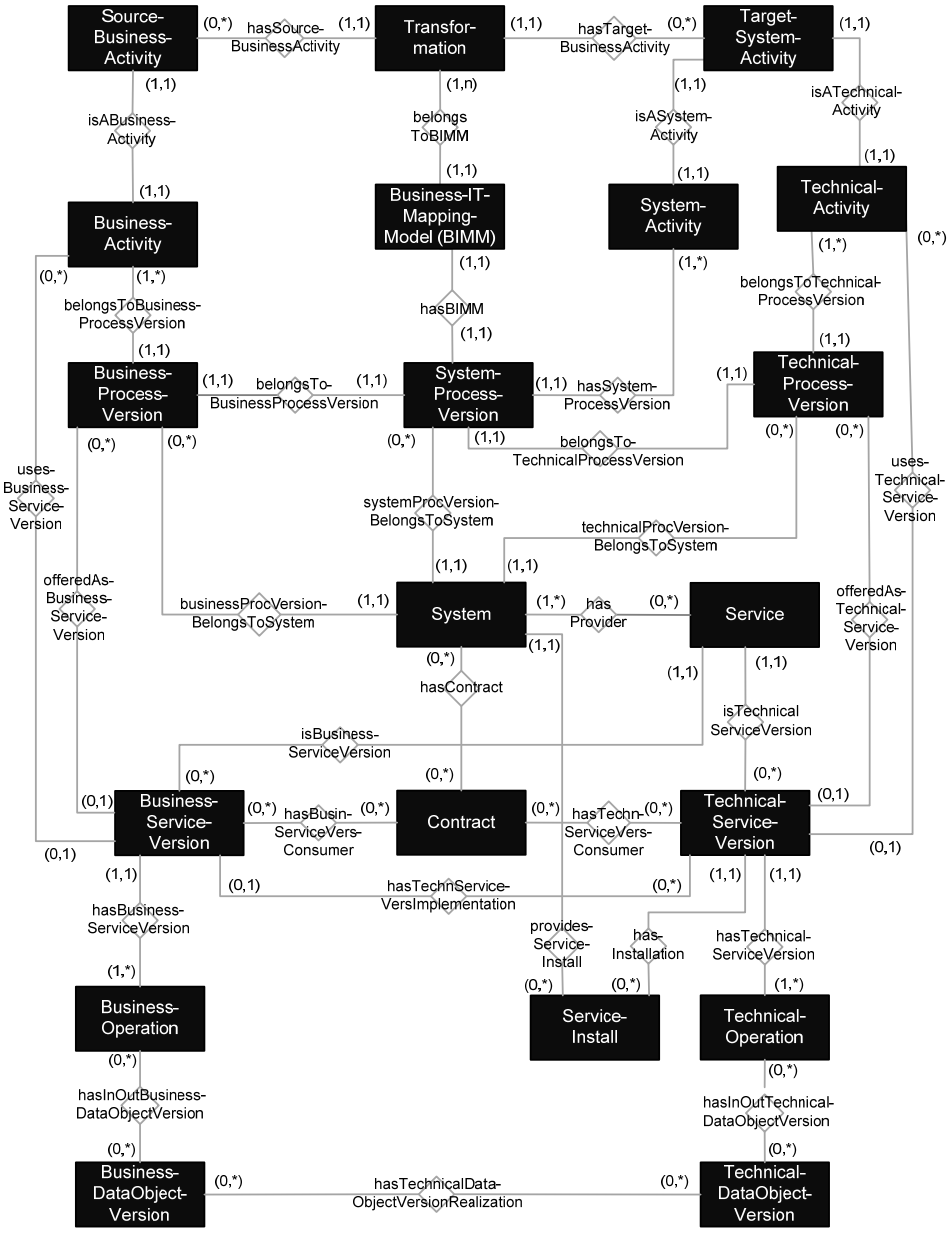

Abbildung 1: Metamodell des SOA-Repository als Entity-Relationship-Diagramm

Die soeben vorgestellten Entitäts- und Beziehungstypen werden wir im Folgenden verwenden, wofür noch die Notation erläutert werden soll: $e \in E$ ist erfüllt gdw. im SOA-Repository die Entität e in der Entitätsmenge E vorhanden ist. Für den Beziehungstyp b zwischen Entitätstyp E und E' mit $e \in E$ und $e' \in E'$ liefert die Funktion $b(e, e')$ wahr gdw. eine entsprechende Beziehung zwischen e und e' im SOA-Repository existiert.

3 Analyse bereits existierender SOA-Repository-Daten (Ist-Analysen)

Ist-Analysen basieren auf bereits im SOA-Repository vorhandenen Daten. Da Contracts jedoch auch in der Zukunft liegende Termine für Beginn und Ende einer Service-Nutzung beinhalten, können dadurch nicht nur bereits bestehende, sondern auch noch zu erwartende Problemsituationen identifiziert werden. Deshalb ermöglichen bereits Ist-Analysen die rechtzeitige Reaktion auf zu erwartende Problemsituationen. In diesem Abschnitt werden exemplarisch einige Ist-Analysen vorgestellt (weitere in [Bu12, Ti10]).

3.1 Technische Service-Version ohne Installation bzw. fachliche Service-Version

Ein sehr einfaches Analyseszenario ist die Identifikation von technischen Service-Versionen, für die keine Service-Installation existiert. Dieser Fall ist interessant, da eine solche technische Service-Version aktuell nicht nutzbar ist. Deshalb muss im Rahmen der SOA-Governance entschieden werden, ob eine entsprechende Service-Implementierung und -Installation durchgeführt werden soll oder die Service-Version nicht benötigt wird, also gelöscht werden kann. Wie alle Analysen basiert die Ermittlung solcher technischer Service-Versionen *TSV* auf den Daten aus dem SOA-Repository:
$TSV = \{tsv \in TechnicalServiceVersion \mid \nexists\, si \in ServiceInstall$ mit $hasInstallation(tsv, si)\}$

3.2 Nicht nutzbare Service-Versionen

Mit ähnlichen Regeln ist basierend auf den SOA-Repository-Daten ebenfalls einfach analysierbar, welche fachlichen und technischen Services aktuell nicht nutzbar sind. So kann zu jedem Service ausgelesen werden, welche BusinessServiceVersion existieren (mittels Beziehung isBusinessServiceVersion in Abbildung 1). In dem in Abbildung 2 dargestellten Beispiel ergeben sich für ServiceX die fachlichen Service-Versionen BSV1, BSV2 und BSV3. Falls einem dieser keine technische Service-Version zugeordnet ist, so ist dieser nicht nutzbar (BSV3 bzw. das unten dargestellte Szenario 3c). Für BSV1 existieren die technischen Service-Versionen TSV1a und TSV1b, für BSV2 existieren TSV2a und TSV2b. Ebenso ist erkennbar, dass z.B. TSV1b keine Service-Installation zugeordnet ist, so dass diese technische Service-Version nicht direkt nutzbar ist (siehe Szenarios 3a und 3b).

Wir betrachten nun, wann eine nicht nutzbare fachliche bzw. technische Service-Version zu Problemen führt und welche Maßnahmen zu deren Lösung ggf. ergriffen werden können. Die Analyse kann hierbei automatisch erfolgen; das Festlegen von Maßnahmen

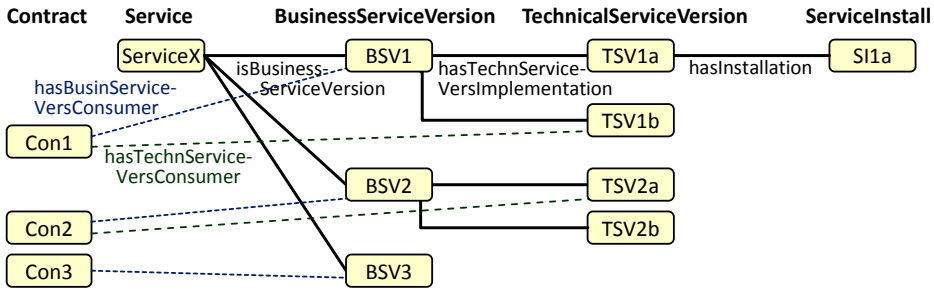

Abbildung 2: Beispieldaten aus dem SOA-Repository zu Contracts, Services, etc.

bleibt selbstverständlich eine intellektuelle Leistung, die z.B. von einem Governance-Gremium übernommen wird. Wir unterscheiden folgende Szenarien:

Szenario 1 (Kein Contract für eine nicht nutzbare fachliche Service-Version): In Abbildung 2 ist BSV2 nicht nutzbar, weil ihm keine Service-Installation zugeordnet ist (über TSV2a oder TSV2b). Angenommen, für BSV2 gibt es keinen relevanten Contract (z.B. weil das in Con2 vereinbarte Nutzungsende in der Vergangenheit liegt), dann besteht kein Problem für die Stabilität der Service-Consumer. Allerdings ist BSV2 nicht nutzbar, so dass die aktuelle Situation verändert werden sollte: Einerseits kann beschlossen werden, dass für TSV2a oder TSV2b eine Service-Installation entwickelt wird. Andererseits kann BSV2 auch für unnötig befunden und deshalb sein „Sun-down" beschlossen werden (inkl. TSV2a und TSV2b). Dies ist z.B. der Fall, wenn sich BSV2 als ungeeignet erwiesen hat, weshalb die zu TSV2a/b gehörenden Service-Installationen bereits abgeschaltet und aus dem SOA-Repository entfernt wurden.

Szenario 2 (Kein Contract für nicht nutzbare technische Service-Version): Gibt es für eine nicht nutzbare technische Service-Version (z.B. TSV1b in Abbildung 2) keinen gültigen Contract, dann besteht wieder kein unmittelbares Problem. Das Governance-Gremium kann entscheiden, ob für TSV1b eine Service-Installation erstellt werden soll (bzw. bereits in Entwicklung ist) oder ob diese technische Service-Version verzichtbar ist, weil sie gegenüber der alternativ verwendbaren TSV1a kaum oder keine Vorteile hat.

Szenario 3 (Contract für nicht nutzbare fachliche bzw. technische Service-Version): Interessanter ist der Fall, wenn die in Abbildung 2 dargestellten Contracts noch nicht abgelaufene Nutzungszeiten beinhalten. Dann existieren relevante Contracts für nicht nutzbare Service-Version. Dieses Szenario unterscheiden weiter nach dem Kriterium, ob alternativ nutzbare Service-Versionen existieren:

Szenario 3a (Technische Service-Version mit nutzbarer alternativer technischer Service-Version): In Abbildung 2 existiert der Contract Con1 für TSV1b, so dass es für TSV1b auch tatsächlich einen Consumer gibt bzw. geben wird. Dies ist problematisch, da TSV1b über keine Service-Installation verfügt. Allerdings gibt es eine weitere technische Service-Version TSV1a, die derselben fachlichen Service-Version zugeordnet ist und zudem nutzbar ist (wg. SI1a). Solche technischen Service-Versionen *TSV* können mittels der Repository-Daten durch die folgenden Kriterien automatisch erkannt werden:

$TSV = \{tsv \in TechnicalServiceVersion \mid$
$\exists\ con \in Contract$ mit $hasTechnServiceVersConsumer(con, tsv) \wedge$
$\nexists\ si \in ServiceInstall$ mit $hasInstallation(tsv, si) \wedge$
$\exists\ bsv \in BusinessServiceVersion$ mit $hasTechnServiceVersImplementation(bsv, tsv) \wedge$
$\exists\ tsv' \in TechnicalServiceVersion$ mit $hasTechnServiceVersImplementation(bsv, tsv') \wedge$
$\exists\ si' \in ServiceInstall$ mit $hasInstallation(tsv', si')$

Als Lösung kann in diesem Szenario von der nicht nutzbaren technischen Service-Version TSV1b zur nutzbaren TSV1a gewechselt werden. Dies ist sinnvoll, wenn TSV1b gegenüber TSV1a kaum Vorteile hat, erfordert aber eine Anpassung von Con1 und der zugehörigen Consumer-Applikation (falls diese bereits realisiert wurde). Alternativ kann auch die Entwicklung einer Service-Installation für TSV1b beschlossen oder auf den Abschluss einer ohnehin stattfindenden Entwicklung gewartet werden. Unterscheiden sich TSV1a und TSV1b nur geringfügig (z.B. andere, aber ähnliche Datenstrukturen für Ein-/Ausgabeparameter), so kann eine Service-Installation für TSV1b auch dadurch realisiert werden, dass SI1a mittels eines Proxy (ESB) gerufen wird, der die Unterschiede zwischen Consumer und Service durch geeignete Transformationen (der Datenstrukturen) ausgleicht. Eine solche Lösung nutzt die durch eine SOA ermöglichte Flexibilität [BBR11].

Szenario 3b (Technische Service-Version ohne nutzbare alternative technische Service-Version) In Abbildung 2 referenziert Con2 die nicht nutzbare technische Service-Version TSV2a. Die einzige ebenfalls zu BSV2 gehörende technische Service-Version TSV2b ist als Alternative nicht nutzbar, da sie über keine Service-Installation verfügt. Solche technischen Service-Versionen *TSV* sind wie folgt erkennbar:
$TSV = \{tsv \in TechnicalServiceVersion \mid$
$\exists\ con \in Contract$ mit $hasTechnServiceVersConsumer(con, tsv) \wedge$
$\nexists\ si \in ServiceInstall$ mit $hasInstallation(tsv, si) \wedge$
$\exists\ bsv \in BusinessServiceVersion$ mit $hasTechnServiceVersImplementation(bsv, tsv) \wedge$
$\nexists\ tsv' \in TechnicalServiceVersion$ mit ($hasTechnServiceVersImplementation(bsv, tsv')$
$\wedge \exists\ si' \in ServiceInstall$ mit $hasInstallation(tsv', si')$)

Selbstverständlich kann auch in diesem Szenario eine geeignete Service-Installation für TSV2a realisiert werden. Soll dies vermieden werden, so sind die Lösungsmöglichkeiten eingeschränkt: Da keine zur selben fachlichen Service-Version gehörende und zudem nutzbare alternative technische Service-Version existiert, bleibt nur der Wechsel zu einer anderen fachlichen Service-Version. Da diese nutzbar sein soll, muss dies in dem in Abbildung 2 dargestellten Beispiel BSV1 sein. Zu BSV1 kann gewechselt werden, indem Con2 und ggf. die Consumer-Applikation entsprechend angepasst werden. Die Veränderungen für letztere sind typischerweise deutlich umfangreicher als bei Szenario 3a, weil sich fachliche Service-Versionen i.d.R. viel stärker unterscheiden als technische.

Szenario 3c (Fachliche Service-Version ohne technische Service-Versionen): Für den Contract Con3 existiert lediglich eine fachliche Service-Version, aber keine korrespondierende technische Service-Version. Dies kann auftreten, wenn die zu realisierende technische Service-Version noch nicht endgültig beschlossen oder spezifiziert wurde. Prinzipiell muss dann auf eine entsprechende technische Service-Version gewartet und

dann der Contract vervollständigt werden. Alternativ kann wie in Szenario 3b zu einer anderen (bereits nutzbaren) fachlichen Service-Version gewechselt werden. Auf die formalen Kriterien zur Erkennung dieses Szenarios verzichten wir aus Platzgründen.

4 Planspiele mit zukünftig möglichen Daten (Impact-Analysen)

Die bisher vorgestellten Analysen betrachten ausschließlich die gegenwärtig im SOA-Repository dargestellte Situation. Die im Folgenden exemplarisch (weitere in [Ti10]) beschriebenen Impact-Analysen betrachten darüber hinaus Auswirkungen möglicher Änderungen in der Zukunft. Mittels solcher Analysen können Szenarien verglichen werden, was dann die Grundlage für Entscheidungen bildet. So kann z.B. analysiert werden, welche Service-Consumer von einer Service-Abschaltung zu einem bestimmten Zeitpunkt betroffen wären, und damit, welcher Abschaltzeitpunkt am günstigsten ist.

4.1 Indirekte Abhängigkeiten (Ripple-Effect)

Die hohe Wichtigkeit von Impact-Analysen ergibt sich, weil von z.B. der Abschaltung einer Service-Installation nicht nur deren unmittelbare Consumer betroffen sind: Diese Consumer sind IT-Systeme (System in Abbildung 1), die selbst Services bereitstellen können. Muss ein solches IT-System verändert werden, weil etwa auf einen alternativen (zu konsumierenden) Service ausgewichen werden muss, so können sich auch Veränderungen an bereitgestellten Services ergeben. Gibt es keinen alternativen Service, dann können Services evtl. überhaupt nicht mehr bereitgestellt werden. In beiden Fällen sind deren Consumer davon betroffen, so dass sich die Änderung über mehrere Stufen zu indirekt abhängigen IT-Systemen „fortpflanzen“ kann (Ripple-Effect).

Solche indirekten Abhängigkeiten lassen sich mit den in Abschnitt 2 beschrieben Repository-Daten automatisch erkennen. In dem in Abbildung 3 dargestellten Beispiel soll die Service-Installation SI1 in naher Zukunft abgeschaltet werden. Sie sei die einzige Realisierung der technischen Service-Version TSV1, wodurch aufgrund der Contracts Con1a, Con1b und Con1c die IT-Systeme Sys1a, Sys1b und Sys1c nicht mehr funktionsfähig sind und deshalb rechtzeitig angepasst werden müssen. Verändern sich dadurch z.B. die von Sys1b angebotenen Services Serv2 und Serv3 bzw. deren technische Service-Version TSV2 und TSV3, so pflanzt sich die Änderung fort: Die hiervon abhängigen Contracts Con2, Con3a und Con3b müssen ebenso angepasst werden, wie die Service-konsumierenden IT-Systeme, die natürlich selbst wieder Services anbieten können, etc.

Die von einer abzuschaltenden oder zu verändernden Service-Installation *si* direkt betroffenen IT-Systeme lassen sich mittels folgendem Kriterium automatisch ermitteln:

AffectedSys(si) = {*sys* ∈ *System* | ∃ *tsv* ∈ *TechnicalServiceVersion* mit
hasInstallation (*tsv*, *si*) für die gilt: ∃ *con* ∈ *Contract* mit
hasTechnServiceVersConsumer(*con*, *tsv*) und *hasContract* (*sys*, *con*) }

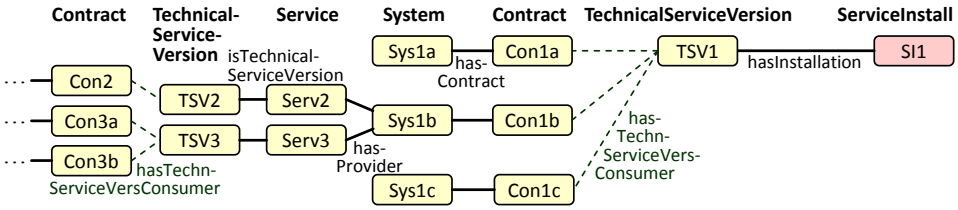

Abbildung 3: Beispieldaten aus dem SOA-Repository mit Ripple-Effect

Indirekt betroffene Systeme lassen sich dann ermitteln, indem man mit der Beziehung *providesServiceInstall* alle von einem System *sys* ∈ *AffectedSys(si)* bereitgestellten Service-Installationen *AffectedServiceInstall* ermittelt und für jedes *si'* ∈ *AffectedService-Install* eine entsprechende Menge *AffectedSys(si')* der nächsten Stufe berechnet, etc.

Wenn man lediglich die Beziehungen zwischen IT-Systemen (System) und Ausprägungen von Services (Service-Install, etc.) betrachtet, dann können ungünstigerweise auch Abhängigkeiten berechnet werden, die in Wirklichkeit überhaupt nicht existieren. Dass die Programmierung eines IT-Systems (irgendwo) einen bestimmten Service aufruft, bedeutet noch nicht, dass dieser für die Bereitstellung jedes angebotenen Services notwendig ist. So sind in dem erläuterten Beispiel das System Sys1b und damit die bereitgestellten Service Serv2 und Serv3 von der abzuschaltenden Service-Installation SI1 abhängig. Fall 1: Angenommen, die Implementierung des konkreten Services TSV2/Serv2 basiert auf TSV1 und SI1. Falls dann die Anpassung der Implementierung von Sys1b zu einer Veränderung der angebotenen Service-Schnittstelle führt, so hat dies Auswirkungen auf den Contract Con2 und die konsumierende Applikation. Fall 2: Angenommen, die Implementierung von TSV3/Serv3 verwendet keinerlei fremde Services oder zumindest nicht den abzuschaltenden SI1. Dann kann die technische Service-Version TSV3 unverändert bleiben, so dass es keinerlei Auswirkungen auf die Contracts Con3a und Con3b gibt. Da die konsumierenden IT-Systeme also nicht betroffen sind, pflanzt sich die Veränderung auf diesem Weg nicht weiter fort. Die in heutigen SOA-Repositories gespeicherten Informationen sind jedoch nicht detailliert genug, um diese beiden Fälle unterscheiden zu können. Deshalb würde eine Analyse stets irrtümlich zu viele Problemsituationen erkennen.

Das diesem Beitrag zugrunde gelegte SOA-Repository (vgl. Abbildung 1) ermöglicht es, die beiden Fälle zu unterscheiden: Durch die Beziehung *usesTechnicalServiceVersion* ist es möglich, exakt diejenigen Prozessschritte (*TechnicalActivity*) eines Geschäftsprozesses zu bestimmen, die wegen eines entsprechenden Serviceaufrufs von einer abzuschaltenden technischen Service-Version abhängig sind. Mittels *belongsToTechnical-ProcessVersion* lässt sich die zugehörige *TechnicalProcessVersion* (der ausführbare Geschäftsprozess / Workflow) ermitteln. Wird dieser wiederum als Service angeboten, so liefert die Beziehung *offeredAsTechnicalServiceVersion* die zugehörige *Technical-ServiceVersion*. Hat schließlich ein IT-System hierzu einen Contract, so ist es tatsächlich von der Änderung betroffen, es liegt also Fall 1 vor. Wir erhalten also das korrekte Analyseergebnis. Dadurch sind sehr viel bessere Planungen und Entscheidungen möglich, als wenn nur Informationen über die insgesamt von einem IT-System konsumierten und bereitgestellten Services im SOA-Repository existieren.

Die für das SOA-Repository benötigten Informationen entstehen bei Verwendung des ENPROSO-Konzepts ohnehin durch die zugrundeliegende durchgängige Modellierung vom fachlichen Geschäftsprozess, über technische Workflows, bis hin zu Services der SOA-Applikationen [BBR12, Bu12]. Da diese Informationen automatisch ins SOA-Repository übertragen werden können, entsteht kaum Mehraufwand für die Datenerfassung. Wird nicht von prozess-orientierten SOA-Applikationen ausgegangen, ist der Ansatz generell auch z.B. auf UML-basierte Modellierung von Services, Service-Implementierungen und deren Consumer übertragbar.

4.2 Analysen möglicher Szenarien für zukünftige Zeitpunkte (Planspiele)

Wie stark die Auswirkungen einer in der Zukunft durchzuführenden Änderung sind, hängt vom genauen Zeitpunkt ihrer Umsetzung ab. Dementsprechend soll der Zeitverlauf in die Analysen einbezogen werden. Indem Analysen für mehrere Zeitpunkte durchgeführt werden, können unterschiedliche Szenarien durchgespielt und analysiert werden. Solche Planspiele möchten wir nun exemplarisch vorstellen.

Angenommen, es soll die aktuell einzige Service-Installation SI0 der technischen Service-Version TSV0 aus Kostengründen in ca. 6 Monaten abgeschaltet werden. Es ist nun zu analysieren, ob dieser Termin aus Sicht des Gesamtunternehmens günstig ist. Als ersten Schritt bietet es sich an, eine graphische Übersicht der Service-Nutzungsverträge (Contracts) im Verlauf der Zeit zu erstellen (vgl. Abbildung 4). Eine solche Darstellung kann aus den Daten des SOA-Repositories automatisch generiert werden. Im Diagramm ist sehr einfach erkennbar, dass aktuell 4 Contracts für TSV0 existieren und dementsprechend 4 IT-Systeme von SI0 abhängig sind. Nach ca. 6, 10, 17 und 24 Monaten laufen Contracts aus. Diese Termine bieten sich als Abschaltzeitpunkte für TSV0 an, da dann jeweils weniger IT-Systeme betroffen sind.

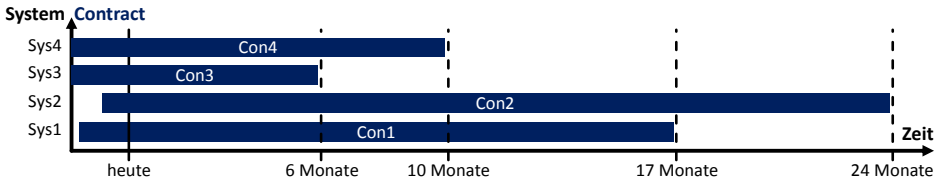

Abbildung 4: Graphische Darstellung der Contracts für TSV0 im Zeitverlauf

Die eben betrachtete Analyse berücksichtigt jedoch noch keine indirekten Abhängigkeiten (Ripple-Effect), so dass die Menge der von der Abschaltung betroffenen IT-Systeme noch unvollständig ist. Der in Abbildung 5a dargestellt Impact-Graph stellt auch solche Abhängigkeiten dar, indem er jede Indirektionsstufe als separate Schale visualisiert. Hierbei werden die zu einem bestimmten Zeitpunkt gültigen Contracts berücksichtigt, bei Abbildung 5a die in 6 Monaten noch gültigen Contracts. Auch dieses Diagramm lässt sich automatisch aus den Daten des SOA-Repositories generieren, wobei die in Abbildung 5b dargestellten Abhängigkeiten angenommen werden: Außer den wegen Con1, Con2, Con4 abhängigen Systemen Sys1, Sys2, Sys4 gibt es noch eine indirekte Abhängigkeit zu Sys5. Das IT-System Sys4 bietet nämlich die Service-Installation SI4 an. Diese implementiert als einzige die technische Service-Version

TSV4, die über Con5 an Sys5 gebunden ist. Eine Abschaltung von SI0 in 6 Monaten würde also die 4 genannten IT-Systeme beeinträchtigen. Mit dieser Information kann auf die Systemeigner zugegangen werden, so dass diese frühzeitig die Folgen der Service-Abschaltung abschätzen und ggf. ihre Systeme geeignet anpassen können.

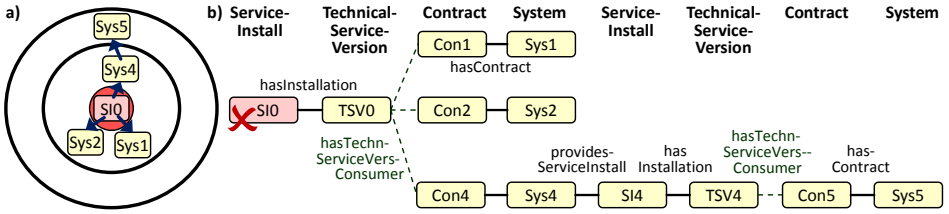

Abbildung 5: a) Impact-Graph und b) Abhängigkeiten von SI0 in 6 Monaten

Erweist sich der Anpassungsaufwand für diese 4 Systeme im Vergleich zu den Betriebskosten von SI0 als zu hoch, so sollte ein längerer Betrieb von SI0 in Erwägung gezogen werden. Als nächster Abschaltzeitpunkt bietet sich heute in 10 Monaten an, weil dann Con4 ausläuft und Sys4 nicht mehr betroffen ist (vgl. Abbildung 4). Dann kann auch das IT-System Sys5 nicht mehr (indirekt) betroffen sein, weil dessen Abhängigkeit ausschließlich über Sys4 bestand. Für die Darstellung des resultierenden Abhängigkeitsgraphen verweisen wir aus Platzgründen ebenso auf [BBTR14], wie für die Analyse der weiteren Abschaltzeitpunkte.

Durch solche Planspiele wird es möglich, abzuwägen, ob eine Service-Abschaltung eher früh umgesetzt werden soll (geringere Betriebskosten), oder ob es aus betriebswirtschaftlicher Sicht Sinn macht, länger zu warten (geringere Migrationskosten). Diese Entscheidung ist von dem zuständigen SOA-Governance-Gremium zu treffen, das durch automatisch aus dem SOA-Repository abgeleitete Informationen unterstützt wird.

5 Diskussion

In der wissenschaftlichen SOA-Literatur werden die hier vorgestellten Themen nicht unmittelbar adressiert (vgl. [BBTR14]). Jedoch gibt es zur Realisierung von SOA-Repositories Standards und Produkte: Das Metamodell des standardisierten Verzeichnisdienstes UDDI [OA02] ist sehr eingeschränkt und sieht auch keine Erweiterungen vor. Außer einigen vordefinierten Abfragen sind keine Analysen vorgesehen. Das Produkt IBM WSRR [Sa07] ermöglicht die Speicherung von technischen und stark eingeschränkt auch fachlichen Objekten. Allerdings ist WSRR bzgl. der Objekttypen und Beziehungen erweiterbar, so dass es als Basis zur Umsetzung des vorgestellten Metamodells verwendbar wäre. Die eher rudimentären, vorhanden Analysen sind um benutzerdefinierte Analysen erweiterbar. Ähnliches gilt für CentraSite Active SOA [Ro06], sowohl was die Objekttypen, die Erweiterbarkeit als auch was die Analysen betrifft. Sowohl Oracle Enterprise Repository [Or08] wie auch HP SOA Systinet [He10] basieren auf UDDI. Beide ermöglichen eine gegenüber dem UDDI-Standard erweiterte Speicherung von Objekttypen, so dass auch fachliche Objekte verwaltet werden können. Die Mächtigkeit der Visualisierung von Abhängigkeitsdiagrammen und der Analysen bleibt weit hinter

den in diesem Beitrag vorgestellten zurück. Das ARIS-Repository [ID08] ermöglicht es die Speicherung von sehr vielen (insb. fachlichen) Objekt- und Beziehungstypen. ARIS unterstützt (eingeschränkt mächtige) Analysen, die auch durch selbst realisierte Analysearten erweiterbar sind (vgl. [Bu12]).

6 Zusammenfassung und Ausblick

Wir haben ein umfassendes Metamodell für ein SOA-Repository vorgestellt, das als Basis für Analyse dient. So kann mit Ist-Analysen geprüft werden, ob aufgrund der gespeicherten Daten mit Problemen (z.B. wegfallende Service-Installation) zu rechnen ist. Mit Impact-Analysen kann simuliert werden, welche Auswirkungen zukünftige Änderungen haben werden, wobei auch die Fortpflanzung von Problemen (Ripple-Effect) berücksichtigt wird. All dies geht deutlich über die bisher in der SOA-Literatur erwähnte bzw. in Standards und Produkten vorgesehene Funktionalität hinaus. Allerdings ermöglicht die Erweiterbarkeit vieler Produkte die Umsetzung der vorgestellten Konzepte. So wurden insbesondere die Analyseverfahren mittels eines technischen Prototyps getestet [Ti10]. Eine vollständige Umsetzung des Metamodells und aller Analysen in einem SOA-Repository-Produkt sowie eine Erprobung des Ansatzes mittels einer Fallstudie wären wünschenswert, konnten bisher aber noch nicht realisiert werden.

Literaturverzeichnis

[Ab74] Abrial, J.R.: Data Semantics. Proc. IFIP Working Conf. Data Base Management, 1974.

[BBR11] Buchwald, S.; Bauer, T.; Reichert, M.: Flexible Prozessapplikationen in Service-orientierten Architekturen - Ein Überblick. EMISA Forum, 31(3), 2011.

[BBR12] Buchwald, S.; Bauer, T.; Reichert, M.: Bridging the Gap Between Business Process Models and Service Composition Specifications. In (Lee et.al., Hrsg): Service Life Cycle Tools and Technologies: Methods, Trends, and Advances. IGI Global, Hershey, 2012.

[BBTR14] Bauer, T.; Buchwald, S.; Tiedeken, J.; Reichert, M.: Erhöhung der System-Stabilität und -Flexibilität durch ein SOA-Repository mit Analysefähigkeiten. HNU Working Paper Nr. 32, Hochschule Neu-Ulm, 2014.

[BTBR10] Buchwald, S.; Tiedeken, J.; Bauer, T.; Reichert, M.: Anforderungen an ein Metamodell für SOA-Repositories. Proc. 2nd Central-European Workshop on Services and their Composition, 2010.

[Bu12] Buchwald, S.: Erhöhung der Durchgängigkeit und Flexibilität prozessorientierter Applikationen mittels Service-Orientierung. Dissertation, Universität Ulm, 2012.

[He10] Hewlett-Packard Development Company: HP SOA Systinet Software Data Sheet, 2010.

[ID08] IDS Scheer: ARIS SOA Architect - Geschäftsprozesse als Grundlage für Service-orientierte Architekturen. IDS Scheer, 2008.

[OA02] OASIS: Universal Description, Discovery, and Integration (UDDI). Version 3.0, 2002.

[Or08] Oracle: Oracle Enterprise Repository and Oracle Service Registry for the SOA-Lifecycle. Oracle Data Sheet, 2008.

[Ro06] Rogers, S.: CentraSite: An Integrated SOA Registry and Repository. Software AG, 2006.

[Sa07] Sachdeva, N.: Customize the WebSphere Service Registry and Repository Governance Profile. IBM developerWorks, 2007.

[Ti10] Tiedeken, J.: Konzeption und Realisierung eines logisch zentralen SOA-Repositories. Diplomarbeit, Universität Ulm, 2010.